05/09/2022 Monday: 1:30PM

2:27min Lecture 69 :- Constructor - default constructor - Constructor Overloading :-

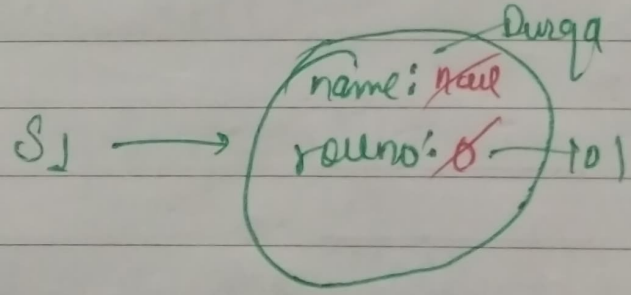
Object Oriented programing: Need Of Constructor.

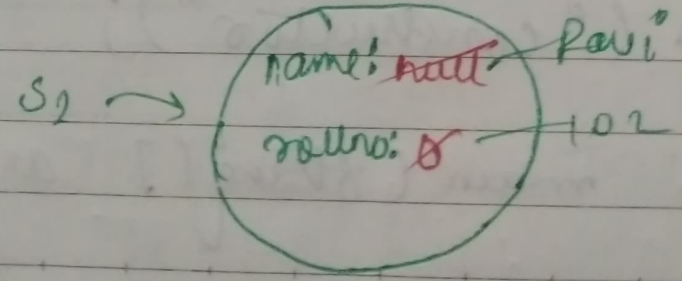
What is Constructor.:-

```
Class Student
{
    String name;
    int rollno;

    Student (String name, int rollno)
    {
        this.name = name;
        this.rollno = rollno;
    }

    P S V main (String[] args)
    {
        Student S1 = new Student ("Durga", 101);
        Student S2 = new Student ("Ravi", 102);
    }
}
```

Ravi Durga 102 101

Constructor





eg: class Student
{
String name;
int rollno;

Student (String name, int rollno)
{
this.name = name;
this.rollno = rollno;
}

public s v main (String[] args)
{
Student s1 = new Student ("Durga", 101);
Student s2 = new Student ("Ravi", 102);
System.out.println (s1.name + ".." + s1.rollno);
System.out.println (s2.name + ".." + s2.rollno);
}
}

o/p: durga .. 101
Ravi .. 102

How many times constructor will executed?

class Test
{
Test ()
{
Sop ln ("constructor");
}

p s v main (String[] args)
{

```
Test t1 = new Test();
Test t2 = new Test();
Test t3 = new Test();
Test t4 = new Test();
}
}
```

Op:- Constructor
Constructor
Constructor
Constructor.

Rules for constructor :-

(1)
```
class Test
{
    Test()
    {
    }
}
```

(2)
```
class Test
{
    void Test()
    {
        sop("Hello");
    }
    p s v main(String[] args)
    {
        Test t1 = new Test();
        Test t2 = new Test();
        t1.Test();
    }
}
```

compiler treats its as a method. but not constructor.

→ Which modifies are applicable for constructors?

public
private
protected
default

eg:
```
class Test
{
    public Test()
    {
    }
}
```

→ Default constructor & prototype :-

default constructor :- The constructor which is generated by compiler is no-arg constructor.

```
class Test
{
    default constructor
}
```

```
class Test
{
    Test(int i)
    {
    }
}
```

prototype of default constructor :-

```
public class Test
{
    public Test()
    {
        super();
    }
}
```

① It is always no-arg constructor
② Access modifier of default constructor

Same as class modifier. [but this rule is applicable only for public, <default>].

(3) It is no-arg call to 'super class/parent class constructor.

Default Constructor Case Study:

Programmer Code	Compiler Code.

① class Test
{
}
.
	class Test
	{
	Test()
	{
	super();
	}
	}

② public class Test
{
}
	public class Test
	{
	public Test()
	{
	super();
	}
	}

③ class Test
{
void Test()
{
}
}
	class Test
	{
	Test()
	{
	super();
	}
	void Test()
	{
	}
	}

④
```
class Test
{
    Test()
    {
    }
}
```

```
class Test
{
    Test()
    {
        super();
    }
}
```
Comp.

⑤
```
class Test
{
    Test(int i)
    {
        this();
    }
    Test()
    {
    }
}
```

```
class Test
{
    Test(int i)
    {
        this();
    }
    Test()
    {
        super();
    }
}
```

⑥
```
class Test
{
    Test(int i)
    {
        super();
    }
}
```
same
```
class Test
{
    Test(int i)
    {
        super();
    }
}
```

* Various Cases of super() and this() :-

Case-1 :-
```
class Test
{
```

```
Test()
{
  sopln("constructor");
  super();
}
```

CE: Call to super must be first statement in constructor.

Case2:-

```
class Test
{
  Test()
  {
    super();
    this();
    sopln("constructor");
  }
}
```

use only in first line simultaneously

CE: call to this must be first line in constructor.

Case3:-

```
class Test
{
  public void m1()
  {
    super();
    sopln("method");
  }
}
```

CE: Call to super must be first statement in constructor.

Super(); — It is used to call super no-arg super class constructor

this(); — It is used to call current class constructor from another constructor

You can call constructor directly from another class only. but not from method.

Note: We can use super() and this() only inside a constructor, but not inside a method.

super()
this()

1. We can use only inside constructor
2. We should use only in first line
3. We can use only one but not both simultaneously.

Difference b/w super(), this() and super, this.

```
class P
{
  String s = "Parent variable";
}
class C extends P
{
  String s = "child variable";
  public void m1()
  {
    System.out.println(s); // child
    System.out.println(this.s); // child
```

```
    sopln (super.s); // Parent
  }
}
class Test
{
    p s v main(String[] args)
    {
        C c = new C();
        c.m1();
    }
}
```

O/p:- child variable
child variable
Parent variable

* If we use this static method then we will get CTE! non-static variable this cannot be used referenced from a static content/area.

	super(), this()	super, this.
①	These are constructor call, to call super class and current class constructor	These are the keywords to refer super class and current class instance members.
②	we can use only in constructor, as first statement only.	We can use anywhere except static area.
③	We can use only one, but not both simultaneously.	We can use any number of times.

Constructor Overloading :

```
class Test
{
    Test (double d)
    {
        this(10);
        sopln ("double-arg Constructor");
    }
    Test (int i)
    {
        this();
        sopln ("int-arg Constructor");
    }
    Test()
    {
        sopln ("no-arg Constructor");
    }
    psvm (String[] args)
    {
        Test t1 = new Test (10.5);
        Test t2 = new Test (10);
        Test t3 = new Test();
    }
}
```

Overloaded constructor

no-arg
int-arg
double-arg

no-arg
int-arg

no-arg.

* Constructor inheritance and Overreding:

```
class P
{
    public void m1()
    {
    }
}
class C extends P
{
    public void m2()
    {
    }
}
C c=new C();
c.m1();
c.m2();
```

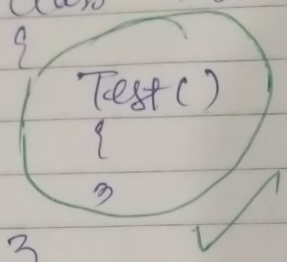
available by default in the case of method.

```
class P
{
    P()
    {
    }
}
class C extends P
{
    C(int i)
    {
    }
}
C c=new C();
```

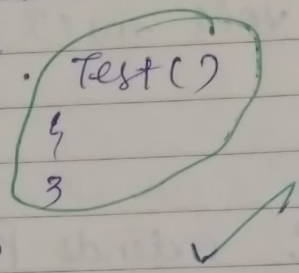
not available by default in the case of constructor

So, inheritance and Overloading Concept is not applicable for constructor.

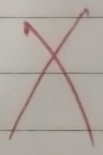
class Test
{
Test()
{
}
}



Abstract class Test
{
Test()
{
}
}



interface Test
{
Test()
{
}
}



Inside interface there is no chance of existing instance variable so no quest of initializn of instance variable, thereby constructor concept not applicable for interfaces.

But inside abstract class instance variable there may be a chance so to perform initializn for that instance variable abstract class can contain Constructor.

Normal Concrete class obviously can contain Constructor.

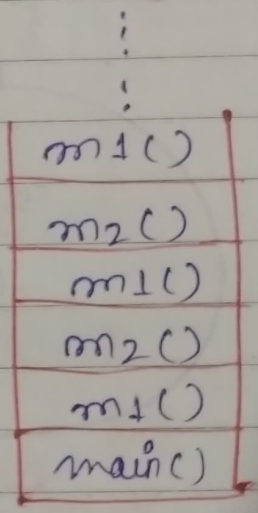
Recursive method and Constructor Calls :-

Method Case :-

```
class Test
{
    p s v m1()
    {
        m2();
    }
    p s v m2()
    {
        m1();
    }
    p s v main(String[] args)
    {
        m1();
        sopln("Hello");
    }
}
```

Op :- Hello.

↳ RE: StackOverflowError



Constructor Case :-

```
class Test
{
    Test()
    {
        this(10);
    }
    Test(int i)
    {
        this();
    }
}
```

```
P {
    s v main(String[] args)
    {
        super("Hello");
    }
}
```

CE: recursive constructor invocation.

* Parent and child class Constructor calls by super.() :-

①

```
class P
{
    P()
    {
        super();
    }
}
class C extends P
{
    C()
    {
        super();
    }
}
```

②

```
class P
{
    P()
    {
        super();
    }
}
class C extends P
{
    C()
    {
        super();
    }
}
```

③

```
class P
{
    P(int i)
    {
        super();
    }
}
class C extends P
{
```

```
C()
{
  super();
}
```

CE: Constructor P in class P cannot be applied to given types.

req: int
found: no arguments.
reason: actual & formal argument lists differ in length.

④

```
class P
{
  P(int i)
  {
    super();
  }
}
class C extends P
{
  C()
  {
    super(10);
  }
}
```

int-argument constructor available.

⑤

```
class P
{
  P(int i)
  {
    super();
  }
```

```
p()
{
}
}
class C extends P
{
}
```

✓

* throws clause with constructors :-

```
import java.io.*;
class P
{
    p() throws IOException
    {
    }
}
class C extends P
{
    C()
    {
        super();
    }
}
```

C?! unreported exception IOException in default constructor.

```
class P
{
    P()  throws IOException.
    {
    }
}
class C extends P
{
    C()  throws IOException / Exception.
    {
     super();
    }
}
```

If the parent class constructor throws any checked exception. then compulsory while writing child class we take care that every child class should throw the same checked exception or its parents.

End of OOPS with constructor revision